Development of Bayesian Networks from Unified Modeling Language Artifacts

Kathryn Blackmond Laskey
Dept. of Systems Engineering
George Mason University
Fairfax, VA 22032-4444
klaskey@gmu.edu

Philip S. Barry
The MITRE Corporation
1820 Dolley Madison Blvd.
McLean, VA 22102-3481
pbarry@mitre.org

Peggy S. Brouse
Dept. of Systems Engineering
George Mason University
Fairfax, VA 22032-4444
pbrouse@gmu.edu

Abstract

This paper examines how Bayesian Networks can be generated from development artifacts intrinsic in the Unified Process. The Unified Software Development Process models the relationship between functional requirements in the Use-Case model. These relationships are expanded in the Analysis Model and a clear mapping to design components is created. By exploiting the relationships intrinsic in the Use-Case Model, as well as the stereotypical classes and temporal ordering found in the analysis model, Bayesian networks that represent system requirements can be generated. These networks can then be used to model and assess the architectural impact of the requirements early in the system lifecycle, thus promoting a more efficient and effective system design process. Formal capture of requirements during design also promotes reusability of design knowledge across similar system.

1 Introduction

Within systems engineering it is well recognized that effective requirements analysis and documentation is essential to the development of high performing and reliable systems [4]. This is equally true in software engineering, where requirements management is generally recognized as a key indicator of organizational maturity [18]. Methodologies for developing and documenting software requirements [e.g., 8,11,19,21] have proliferated in the literature and have been employed with varying degrees of success.

The Unified Modeling Language (UML) [5] is rapidly becoming the de facto standard for analysis and design within the software development community. UML provides a diagrammatic approach to describing user requirements, which begins with Use-Cases and then leads into more formal specification using stereotypical classes in an analysis model. Key elements of these artifacts form the basis for architectural views into the system as well as providing the groundwork for design, implementation and validation and verification.

From an architecture viewpoint, all Use-Cases are not equally important, necessitating the need to develop a scheme to prioritize the most significant Use-Cases. This need comes from the requirement to focus the design effort and often stage the design and implementation of key features to meet release schedules and budgetary constraints. In practice, determining which Use-Cases are architecturally significant and how they may relate to other Use-Cases as well as additional system requirements is usually determined by heuristic methods. We assert by exploiting the implicit structure in UML diagrams an algorithmic and mathematically sound method can be introduced to understand the importance of use requirements represented as Use-Cases as well as the system requirements they imply.

2 Use-Cases: The Requirements Side of UML

According to Ivar Jacobson [12] "it is absurd to believe that the human mind can come up with a consistent and relevant list of thousands of requirements of the form "The system shall…". The futility of direct specification has led to a search for more human-friendly ways to develop system requirements. A highly promising and popular approach is to develop requirements from a set of Use-Cases. Use-Cases represent functionality of the system in terms of prototypical problems or processes the system is expected to execute. Thus, a Use-Case represents one distinct way of using the system from a given external agent or actor's view. Use-Cases are generally specified using a graphically based methodology for defining the actions the system is to perform. Analyzing the functionality needed for successful execution of the Use-Case then derives system requirements.

A Use-Case specifies a primary sequence of actions the system is to perform, and may also include excursions from the main sequence. Use-Cases have operations and attributes. Thus, a Use-Case description can include state chart diagrams, collaborations and sequence diagrams. Use-Case attributes represent the values that a Use-Case instance uses and manipulates. Use-Case instances do not interact with other Use-Case instances because by design Use-Case models should be simple and intuitive. The flow of events and special requirements are captured in special textual descriptions.

2.1 Structuring the Use-Case Model

System designers can specify relationships between Use-Cases. These relationships can be used to form interesting structures that can provide evidence to the importance of requirements based upon their relationships to other requirements. There are three relationships used to model Use-Cases: generalization, extension and inclusion.

- Generalization between Use-Cases is a kind of inheritance. Use-Cases can perform all behavior described in the generalizing Use-Case.
- Extension models additions to a Use-Case's sequence of actions. These additions are contingent upon some conditions being satisfied.
- Inclusion between Use-Cases models the situation where the including Use-Case cannot function without the included Use-Case.

3 Analysis Models

Within the Unified Software Development Process [12], analysis is used to address previously unresolved issues by analyzing the requirements in depth. Key to analysis is the development of the analysis model, which yields a more precise specification of the requirements than resulted from the Use-Case model. As the analysis model is described in the language of the developers it can introduce more formalism and can be used to reason about the internal workings of the system. The analysis model can be considered a first cut at a design model and is an essential input to system design and implementation.

The analysis model makes use of a number of diagrams. Of particular interest is the collaboration diagram, which describes how a specific Use-Case is realized and performed. The diagram is drawn in terms of stereotypical analysis classes and their instantiated analysis objects. These objects focus on handling functional requirements and postpone the consideration of nonfunctional requirements by designating them as special requirements.

In the analysis phase of the Unified Development process, there are three stereotypical classes: boundary classes, entity classes and control classes. Boundary Classes are used to model interaction between the system and its actors, e.g., information input and output and can be used to clarify system boundaries. Entity Classes are used to model information that is long-lived and persistent, which may have associated behaviors and a logical data structure. Control Classes are used to model coordination, sequencing, transactions and control of other objects.

4 Bayesian Networks for Requirements

Key to fully understanding the requisite functionality of a system is the understanding of the relationships between the system requirements. In addition, if a system requirement is the result of user needs, other system requirements are often generated. This expansion of system requirements contingent upon an agent's goals is often referred to as allocation and flowdown [9].

We suggest that requirements engineering can benefit from a computational mechanism to represent of system requirements to each other and to the user requirements that engender them. To achieve this the relationship we define an abstract structure of interrelated system requirements called a system requirement web (SRW). A SRW is a directed graph in which the nodes represent system requirements and the edges represent relationships between requirements that we call weak implication. We say that one node weakly implies another node if it is more likely that the requirement represented by the second node is needed if the first one is. User requirements can trigger a flow of weak implication within a SRW, providing a model for allocation and flowdown of requirements.

Previous work by the authors has demonstrated that Bayesian networks provide a natural computational model for SRWs [2,3]. A Bayesian network is both a structured representation for knowledge about the interrelationships among uncertain variables and a computational architecture for reasoning about these variables. A Bayesian network is a directed graph composed of nodes and arcs. The nodes represent variables whose value is uncertain and the arcs represent dependency relationships between the variables. In our application, the uncertain variables represent requirements and the arcs represent weak implication. A requirement can be in one of two states: *implied* or *unimplied* (the semantics of these terms is described below). Conditional probabilities are used to model the strength of the relationship between the associated variables.

As a computational architecture, a Bayesian network allows the user or application to declare "evidence" on some of the nodes and, through a process called "evidence accumulation," compute revised probabilities for all other nodes in the network. In the present application context, user requirements can be declared as "evidence" in an SRW represented as a Bayesian network. This evidence propagates through the SRW via the weak implication links, resulting in updated probabilities for other requirements in the SRW.

Not only are Bayesian networks a useful visual metaphor to aid in the construction of SRWs, they also appropriately represent a semantics appropriate to the problem at hand. At any given moment in the requirements definition process, the currently operating Bayesian network represents a set of requirements, some declared as implied, some undeclared, and possibly others declared as unimplied. After evidence accumulation, each node in the network carries an associated probability that it is implied given the current set of declared evidence. All requirements declared as implied or unimplied have probabilities of unity or zero, respectively. All undeclared requirements have intermediate probabilities. Mathematically, the belief propagation algorithm calculates for each requirement the probability it is implied given the current declared set of implied and unimplied requirements. Semantically, then, the current probability that a requirement is implied represents the probability that this requirement is a necessary element of functionality in a system that satisfies the functionality represented by the declared set. Declaring new requirements as either implied or unimplied initiates a new evidence accumulation cycle that results in revised probabilities for undeclared requirements. This formalism is a powerful design mechanism for exploring the interrelationships among requirements.

Especially important to the requirements definition process is the ability to identify and adjudicate conflicts among requirements. An attempt to specify incompatible requirements results in an error condition in the associated Bayesian network. Evidence accumulation generates an error condition when the pattern of evidence is logically impossible according to the model. When inconsistencies are discovered, this means the associated requirements are incompatible given the constraints represented in the Bayesian network. When this occurs, the software engineer can explore ways to adjudicate the conflict by use of a feature of Bayesian networks called “soft evidence.” Soft evidence allows the engineer to declare varying degrees of strength with which a given user requirement is implied. Requirements deemed essential can be declared as hard evidence; requirements deemed less than essential can be declared as soft evidence with strengths related to their perceived importance. If outright conflicts are discovered, the offending declarations can be relaxed to soft evidence. Varying the strengths of the soft evidence declarations could be an important sensitivity analysis tool in the design process. Conflict metrics [13, 16] can be used to identify conflicts that fall short of outright inconsistencies and to find clusters of requirements responsible for the conflict.

In previous work [2], it became apparent that the behavior of large SRWs was often difficult to predict and test. We therefore broke up the larger SRWs into manageable SRW fragments that corresponded to a natural decomposition of the domain of interest. By combining these fragments, different views into the domain could be created, emphasizing particular aspects that were important to the analyst. The process for combining the SRW chunks was labeled gluing. Laskey and Mahoney took a similar approach by using network fragments to model situation assessment [17].

5 Mapping Use-Case Models to Bayesian Networks

As discussed, Use-Cases represent the external function view of user requirements and collaboration diagrams specify the top-level system functionality. Requirements elicitation involves an element of uncertainty that the actual need has been captured. Further uncertainty is introduced when we relate the requirements and then transform them into models such as collaboration diagrams. Representing this uncertainty becomes very important when there the sheer number of interrelated and potentially conflicting requirements overwhelms the ability of the unaided human mind. A Bayesian network explicitly models this uncertainty between the requirements as represented by Use-Cases and elements of collaboration diagrams. Contingent upon the introduction of evidence such as the importance of a given actor, a quantitative assessment can be made as to how strongly we believe the requirement is indicated, or, in the parlance of SRWs, implied. We therefore see the ability to translate Use-Case models and their associated collaboration diagrams into Bayesian networks as an important potential advance in software engineering. The SRW provides a way of understanding the degree to which each of the specified requirements is implied by the information encoded in the user requirements. This ability to represent degrees of implication is key to the identification and effective management of conflicts among requirements.

5.1 Generalization

A generalized Use-Case contains common functionality that is allocated to all specializing Use-Cases. Mapping the generalization relationship to Bayesian network fragments is straightforward. Consider Figure 5.1, where Use-Case A is a generalization of Use-Case A1 and Use-Case A2. We view this as functional requirements A1 and A2 being specializations of functional requirement A.

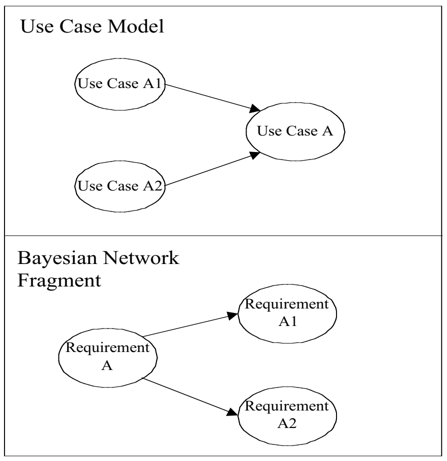

Figure 5.1 Generalization Mapping

This translates into a Bayesian network with an analogous structure. The direction of implication flows from A to A1 and A2 reflecting a top-down decomposition. This indicates that one is more likely to encounter the general case than the specific requirement. Thus:

P(A) = prior
P(A1 | A) = P(A |A1)P(A1) / P(A)
P(A2 | A) = P(A |A21)P(A2) / P(A)

5.2 Inclusion

Inclusion models the situation in which a Use-Case is composed of a number of sub-Use-Cases. In the case of inclusion, the top-level Use-Case cannot execute without the execution of one of the sub-Use-Cases. To see how this can be translated to Bayesian Network fragments, consider Figure 5.2. Use-Case A is related to Use-Case A1 and Use-Case A2 by an inclusion relationship. It is more likely that the need for Use-Case A would be elicited in a requirements generation activity, so we draw the implication arrow from Use-Case A to Use-Case A1 and Use-Case A2. Thus, the probability assignments for the nodes are as before:

P(A) = prior
P(A1 | A) = P(A |A1)P(A1) / P(A)
P(A2 | A) = P(A |A21)P(A2) / P(A)

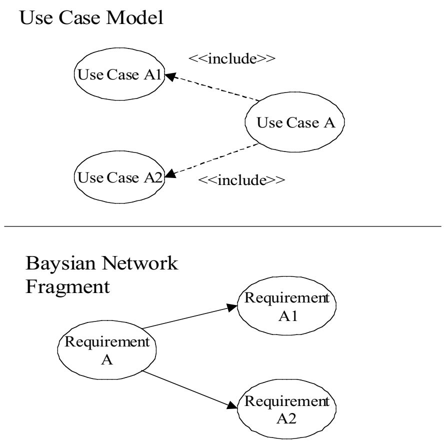

Figure 5.2: Inclusion Mapping

5.3 Extension

Extension models the case where a given Use-Case will branch into additional behavior given the satisfaction of some condition or conditions. In the case of extension, the first Use-Case does not need the additional Use-Case to execute. The second Use-Case represents exceptional behavior if conditions are met.

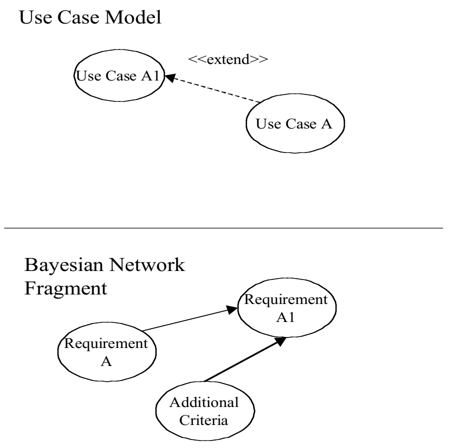

Figure 5.3: Extension Mapping

Consider the general case shown diagrammatically in Figure 5.3. Use-Case A is extended by Use-Case A1. This models the situation where some additional criterion triggers Use-Case A1 after Use-Case A executes. The additional criterion is described in the textual flow of events write-up. This situation is modeled as Requirement A1 implied by Requirement A. The additional criterion is modeled as another requirement node. The direction of implication is from the additional criteria (AC) to requirement A1. Thus:

$$P(A1 \mid A, AC) = \frac{P(A|A1, AC)P(A1|AC)}{P(A|AC)}$$

6 Exploiting the Analysis Artifacts

The analysis process is a refinement and a specification of the Use-Cases into a more formal language. For each Use-Case, an analysis model can be created which specifies in generic terms the system functionality. Analysis models are formed from the instantiation of the stereotypical classes.

6.1 Collaboration Diagram Mappings

A collaboration diagram is the functional specification of an individual Use-Case using instantiated objects from the stereotypical classes, shown in Figure 6.1. The collaboration diagram relates the objects by links that represent messages sent between the objects. The links are numbered, representing a rough ordering. The messages model requests for services from the sending object to the receiving object. These messages provide a finer view of the functionality offered by a given object.

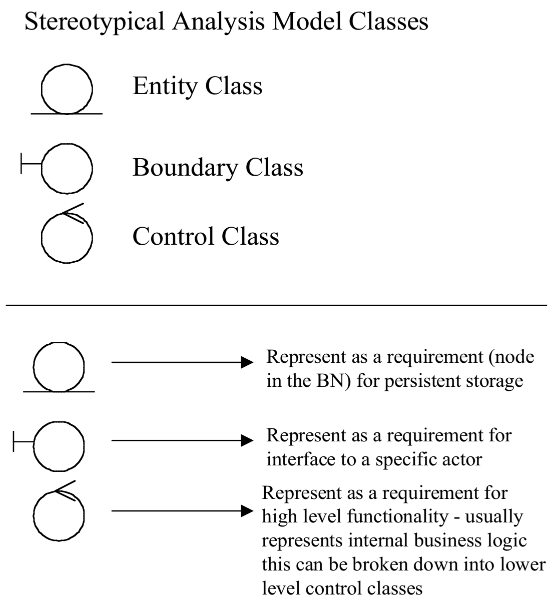
Stereotypical Analysis Model Classes
Entity Class
Boundary Class
Control Class
Represent as a requirement (node in the BN) for persistent storage
Represent as a requirement for interface to a specific actor
Represent as a requirement for high level functionality - usually represents internal business logic this can be broken down into lower level control classes

Figure 6.1: Stereotypical Analysis Classes

Collaboration diagrams are readily mapped into network fragments. Let a collaboration diagram D consists of three stereotypical classes **B**, **C** and **E** for Boundary, Control and Entity. Let function I be an instantiator, which can create objects instantiated from these classes. For example, I(**B,** d) = B would indicate that a boundary object has been created in a way that depends on additional data d**.** Let **O** consist of all of the objects in the collaboration diagram that have been instantiated. Thus, **O = {** I(**B, d**), I(**E, d**), I(**C, d**) **}** where **d** = {d_1, d_2, $d_3 \ldots d_n$} and O $\in$ **O.** With these assumptions, several mapping rules can be readily written.

Rule 1:
If I(**B,** d_i) = B, we write a requirement for actor interface of type **B** which interacts with the actor for data of type d_i. This is represented as a like named node in SRW network fragment **N.**
Rule 1.1:
If Actor A sends a message of type m_i to B, we write a requirement for functionality m_i from object B. This is represented as a like named node M in the BN with an arc drawn from B to M.
Rule 1.2:
We model the Actor A as evidence for requirement M.
Rule 2:
If I(E, d_i) = E, we write a requirement for persistent storage of type E containing information di. This is represented as a like named node in N.
Rule 2.1:
If a message m_i is sent to an entity object E from another object O, we write a requirement for functionality M from object E. This is represented as a like named node M in the BN with an arc drawn from E to M. We further draw an arc from the like named node in the BN that represents O to M.
Rule 3:
If I(C, d_i) = C, we write a requirement for functionality of type C acting on information di. This is represented as a like named node in N.
Rule 3.1:
If a message m_i is sent to a control object C from another object O, we write a requirement for functionality M from object C. This is represented as a like named node M in the BN with an arc drawn from C to M. We further draw an arc from the like named node in the BN, which represents O to M.

6.2 Prior and Conditional Probability Assignment

A key step in the process of building a Bayesian network model is assigning the local probability tables associated with the nodes. The ideal situation would be to have frequency data from which to estimate likelihoods. In requirements engineering, it is rarely the case that such data is available.

In [2, 3] we found that heuristic probability assignment provided good results. We used heuristics both for the specification of probability tables in individual SRW fragments and the combination of tables for common nodes when fragments were combined. Specifically, two types of influence combination methods were used, a linear additive heuristic and the “noisy or”. The linear additive heuristic is used in the instance where an individual requirement is indicated by a group of two or more different requirements in combination (i.e., the weak implication is a “and” relationship). Taking away any one of the requirements in the parent group undermines support for the requirement they together imply. The “noisy or” model was used in the case where each parent alone weakly implies the associated requirement (i.e., the weak implication is an “or” relationship). A good discussion of the “noisy or” distribution is provided in [14].

A concern in the automated generation of Bayesian network fragments is the explicit introduction of conditional independence between requirements, particularly when translating collaboration diagrams. For example, by rule 1.1 specified in Section 6.1 above, if an actor sends a message to a boundary object B, that becomes a requirement indicated by an arc from requirement for the boundary object to the requirement for the functionality represented by the message, say M1. If a second message is sent to B, a second requirement is indicated and an arc is drawn from B to the new requirement M2. This is modeled as a conditional independence relationship in the Bayesian Network, where both the probability that M1 is really needed and the probability that M2 is needed are dependent upon the probability that B is indicated, but are independent of each other given B. Should it be known with certainty whether B is actually indicated, any additional evidence for M1 will not effect the certainty of M2 or vice versa.

In general, we believe that we can make the assumption of conditional independence because we are taking advantage of the precedence relationships between the stereotypical classes and their associated functionality. In the example above, if we know with certainty that a boundary object is not required, both M1 and M2 become (absent other evidence) improbable and independent of each other. That is, evidence for M1 would not affect the probability that M2 is implied, absent the existence of a requirement relating the two. Conversely, if we know that a boundary object is required, M1 and M2 become probable and independent of each other. Again, evidence that M1 is actually not required would not affect the probability that M2 is required if that evidence did not cast doubt on the requirement for the boundary class. Rather, such evidence would indicate the existence of some alternate way to satisfy the role that functionality represented by M1 plays in the boundary class.

7 A Mapping Example

To illustrate the ideas discussed in the previous section, consider the following example. Suppose a requirement elicitation activity has determined that there is a need to develop an application that allows an individual to purchase products over the Internet. A number of functional requirements have been defined, such as log on to the site, browse products, select products, etc. Table 7.1 shows the list of requirements necessary for this system.

• Provide Index Help • Log on • Browse products • Browse with search criteria • Select products • Display selected products	• Add selected products • Remove selected products • Provide On-line Help • Provide FAQs Purchase Products • Purchase using Credit Card • Purchase Products

Table 1: On-Line Product Purchase Functionality List

The decision is made to refine the textual descriptions of functional requirements by creating a top-level Use-Case model, provided as Figure 7.1. In the figure, two actors are identified, namely the buyer of the product and the company computer, named pbdog.com. The Figure shows the association between the Use-Cases and the the actors, as well as the relationships of generalization and inclusion. In particular, the requirement of *Browse w/Search Criteria* has been identified as a generalization of browse products, *Provide On-Line Help* generalizes *Provide FAQs* and *Provide Index Help* and *Purchase Product* is a generalization of *Purchase Using Credit Card.* The inclusion relationship is illustrated by the functionality of *Select Products* requiring *Display Selected Products*, *Add Selected Products* and *Remove Selected Products*.

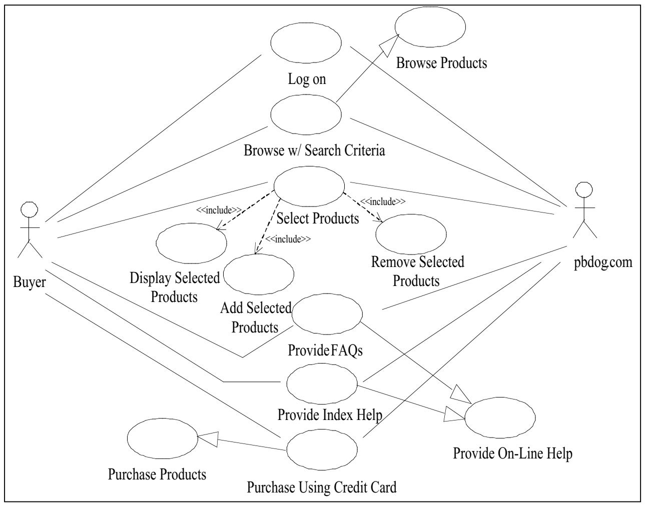

Figure 7.1: Top-Level Use-Case Model

Figure 7.2 represents a collaboration diagram for the Use-Case *Purchase Using Credit Card* developed with stereotypical analysis classes. Notice the addition of messages from the actor, as well as between the stereotypical classes. These will form the basis for the functionality specified in the Bayesian network. While a loose temporal ordering is suggested by the numbering of the messages, sequencing requirements are not explicitly modeled with this diagrammatic technique but rather are shown by interaction diagrams, which are not examined here.

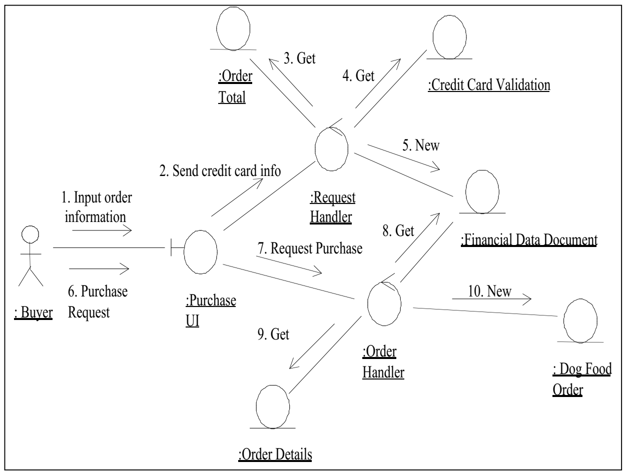

Figure 7.2: Collaboration diagram with Stereotypical Classes

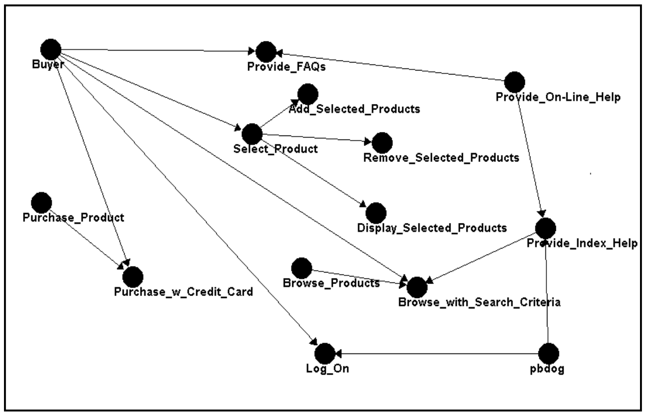

Figure 7.3: Bayesian Network from Use-Case Model

Once the Use-Case models and the collaboration diagrams have been created, it is straightforward to create the structure of the Bayesian Network using the rules described in Sections 5 and 6 above. Figure 7.3 represents the Bayesian Network constructed from the Use-Case model shown in Figure 7.1. Notice how the conditional independence has been directly modeled by application of the rules as stated. For example, *Add_Selected_Products* is conditionally independent of *Remove_Selected* Product given the requirement for *Select_Product*. This is reasonable from a requirements standpoint, for if it is known that functionality *Select_Product* is definitely not needed, any evidence for *Add_Select_Products* provided by *Remove_Selected_Product* can be reasonably discounted. Similarly, if *Select_Product* is definitely indicated, it is reasonable to state that the evidence for *Add_Select_Products* is compelling and we can reasonably disregard information from *Remove_Selected_Product.*

Figure 7.4 represents a Bayesian network fragment developed to represent the collaboration diagram shown in Figure 7.2. For each Use-Case represented in Figure 7.1, a collaboration diagram was developed. These diagrams can then be glued into the higher level Bayesian network, wholly replacing the Use-Case node from which it has been derived. As mentioned, the collaboration diagrams contain common functionality. The common nodes are identified and used as the basis for gluing the fragments. Conditional probabilities are reallocated accordingly. The glued Bayesian network then more truly represents the relationships between system functionality based upon the belief that the user requirements are truly needed, which are contingent upon the importance of the user. Implementation insight for pasting hierarchical Bayesian networks is provided in the object oriented Bayesian Network literature [15, 20].

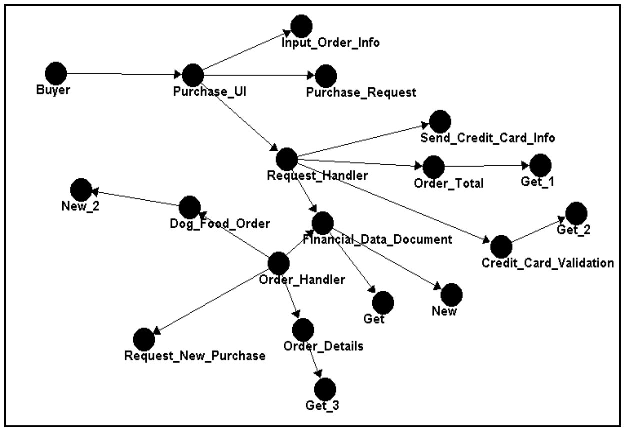

Figure 7.4: Bayesian Network Fragment from Collaboration Diagram

8 Implementation Architecture

The real power of this approach is transparent transformation from UML diagrams into Bayesian Network fragments that are then glued together with minimal user intervention. At the time of this writing, we are focusing on implementing the building blocks for this transparent translation. We are using Rational Rose^{TM} to develop Use-Case models and collaboration diagrams in UML. These models are saved in *.mdl format, which is human readable text. A parser has been written which extracts the relevant information and sends it to an expert system that applies rules discussed in Sections 5 and 6. The expert system, written using Jess [10], outputs text files in Bayes Interchange Format (BIF) which represent Bayesian Network fragments.

Currently, both JavaBayes [7] and the BOSH prototype can read the XML BIF files. JavaBayes is used to visually inspect and debug the network fragments. The BOSH prototype is used to glue the network fragments together, as well as interpret the effect of nonfunctional requirements and other evidence on the glued network. BOSH, introduced in [1], is a prototype agent-based expert system that infers system requirements given evidence from a user requirements specification. BOSH uses APIs from both JavaBayes and JESS.

We are beginning work on two improvements that will generalize the applicability of the prototype. First, the XMI toolkit from IBM [6] provides an API that will read Rational Rose files and translate them into XMI format. We are rewriting the parser to read XMI files, so the system will work with any software that can output XMI as opposed to just one product. The second improvement is more significant, and involves totally encapsulating the implementation of the Bayesian network within the UML diagrams.

9 Summary and Outlook

The Unified Software Development process provides a traceable methodology for system development, iteratively improving on each of the specified software artifacts. In this paper we have suggested that augmenting the requirements specification process with Bayesian Networks provides an algorithmic approach to modeling the relationships between user and system requirements, as well as determining their architectural significance. We assert that a fuller understanding of the requirements space allows for more effective design and implementation decisions as well as project planning.

One of the challenges in requirements engineering is to confirm functional requirements with the user and to then put those requirements in a form that can be used by the designer. This transformation from functional requirements to formal requirements usually introduces errors. After problem analysis and the creation of Use-Cases, which are validated with the user, a transformation must occur to a formalism that is not easily validated with the user. If the analyst is given a tool such as we have suggested herein, we believe this transformation will result in considerably less error.

There are significantly more diagrams and views provided by UML than discussed in this paper. For example, temporal considerations, which may be found in an interaction diagram, have yet to be examined. Similarly, we have not looked at nonfunctional requirements that might be found in deployment diagram. The full integration of these additional types of requirements using Bayesian Network fragments is an exciting area of research that is yet to be explored.

References

[1] Barry, P., *An Agile Approach to Requirements Modeling*, Ph.D. Dissertation, George Mason University, Fairfax, Virginia, May 1999.

[2] Barry, P. and Laskey, K. "An Application of Uncertain Reasoning to Requirements Engineering". "Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence", Morgan Kaufmann Publishers, Inc., San Francisco, 1999.

[3] Barry, P., Laskey, K. and Brouse, P. "Modeling the Requirements for Enterprise Control Systems", Proceedings of the DARPA-JFACC Symposium on Advances in Enterprise Control, November 1999.

[4] Blanchard, B. and Fabrycky, W., *Systems Engineering and Analysis, Third Edition.* Prentice Hall, Upper Saddle River, NJ, 1998.

[5] Booch, G., Rumbaugh, J. and Jacobson, I. *The Unified Modeling Language User Guide.* Addison-Wesley Publishing Company, Reading, MA, 1999.

[6] Brodsky, S., " XMI Opens Application Interchange", http://www-4.ibm.com/software/ad/standards/xmiwhite0399.pdf.

[7] Cozman, Fabio. JavaBayes source code and documentation available http://www.cs.cmu.edu/~javaBayes/index.html/.

[8] Darimont, R. and van Lamsweerde, A. "Formal Refinement Patterns for Goal-Driven Requirements Elaboration", Proceedings of the Fourth ACM Symposium on the Foundations of Software Engineering (FSE4), San Francisco, Oct. 1996.

[9] Dorfman, M. "Requirements Engineering", in *Software Requirements Engineering, 2nd Edition*, Thayer, R. and Dorfman, M. editors, IEEE Press, Los Alamitos, CA, 1997.

[10] Friedman-Hill, Ernest. JESS source code and documentation available at http://herzberg.ca.sandia.gov/.

[11] Jackson, M.A. *System Development.* Prentice Hall Inc., Englewood Cliffs, NJ, 1983.

[12] Jacobson, I., Booch, G. and Rumbaugh, J. *The Unified Software Development Process.* Addison-Wesley Publishing Company, Reading, MA, 1999.

[13] Jensen, F.V., Chamberlain, B., Nordahl, T. and Jensen, F. "Analysis in HUGIN of Data Conflict" . "Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence", Elsevier Science Publishing Company Inc., New York, NY 1991.

[14] Jensen, F.V. *An Introduction to Bayesian Networks*. Springer-Verlag New York, 1996.

[15] Koller, D. and Pfeffer, A. "Object-Oriented Bayesian Networks". "Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence," Morgan Kaufmann Publishers, Inc., San Francisco, 1997.

[16] Laskey, K. "Conflict and Surprise: Heuristics for Model Revision "Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence", Morgan Kaufmann Publishers, Inc., San Mateo,California 1991.

[17] Laskey, K. and Mahoney, S., "Network Fragments: Representing Knowledge for Constructing Probabilistic Models". "Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence," Morgan Kaufmann Publishers, Inc., San Francisco, 1997

[18] Paulk, M., et.al., *The Capability Maturity Model Guidelines for Improving the Software Process.* Addison-Wesley Publishing Company, Reading, MA, 1995.

[19] Peterson, J.L., *Petri-Net Theory and the Modeling of Systems.* Prentice-Hall, Englewood Cliffs, NJ, 1981.

[20] Pfeffer, A., Koller, D., Milch, B. and Takusagawa, K. "SPOOK: A System for Probabilistic Object-Oriented Knowledge Representation". "Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence," Morgan Kaufmann Publishers, Inc., San Francisco, 1999.

[21] Rumbaugh, J. et. al., *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, NJ, 1991.